Analysis of System Overhead on Parallel Computers

Roberto Gioiosa Fabrizio Petrini Kei Davis
Fabien Lebaillif-Delamare
Performance and Architecture Laboratory (PAL)
Computer and Computational Sciences (CCS) Division
Los Alamos National Laboratory, NM 87545, USA
{gioiosa,fabrizio,kei,fabien}@lanl.gov

Abstract

Ever-increasing demand for computing capability is driving the construction of ever-larger computer clusters, typically comprising commodity compute nodes, ranging in size up to thousands of processors, with each node hosting an instance of the operating system (OS). Recent studies [1, 4] have shown that even minimal intrusion by the OS on user applications, e.g. a slowdown of user processes of less than 1.0% on each OS instance, can result in a dramatic performance degradation—50% or more—when the user applications are executed on thousands of processors.

The contribution of this paper is the explication, and demonstration by way of a case study, of a methodology for analyzing and evaluating the impact of the system (all software and hardware other than user applications) activity on application performance. Our methodology has three major components: 1) a set of simple benchmarks to quickly measure and identify the impact of intrusive system events; 2) a kernel-level profiling tool *Oprofile* to characterize all relevant events and their sources; and, 3) a kernel module that provides timing information for in-depth modeling of the frequency and duration of each relevant event and determines which sources have the greatest impact on performance (and are therefore the most important to eliminate).

The paper provides a collection of experimental results conducted on a state-of-the-art dual AMD Opteron cluster running GNU/Linux 2.6.5. While our work has been performed on this specific OS, we argue that our contribution readily generalizes to other open source and commercial operating systems.

Keywords: Parallel Computing, Operating System Noise, Linux, Performance Evaluation, Clusters of Workstations.

1 Introduction

The largest parallel computers based on cluster architectures currently deliver tens of teraflops; one petaflop performance is expected within the decade. Some of these clusters have thousands of processors, such as Livermore's Thunder, Sandia's Cplant, Virginia Tech's Terascale Cluster, and ASCI's Lightning. Such machines are very expensive and are intended to solve as large instances as possible of various scientific applications. Much effort is expended to optimize the applications that run on these machines; as the applications improve and the machines get larger the impact of the system behavior on application performance begins to exceed that of sub-optimalities in the application. In other words, with increasing scale the system is (unnecessarily) becoming the primary bottleneck. The source of application performance degradation is system *noise*—minimal interruptions of application execution by such activities as hardware interrupts. Our goal is to show how this impact may be characterized and quantified, and ultimately eliminated or ameliorated.

1.1 Impact of System Noise at Scale

As will be shown, for a typical one- or two-processor compute node noise events typically result in an insignificant 1-2% slowdown in application execution speed. To see why this is greatly magnified as the number of processors increases requires only the understanding of a very simplistic model of the behavior of common parallel applications. In this model a large data structure is equally distributed over the processors of a cluster, with each processor performing essentially the same computations on its portion of the data. At regular intervals the processors perform a global synchronization/communication (to exchange data at the boundaries of their portions of the global data structure, for example). Thus the rate of overall progress is limited by the processor slowest to complete its computational phase. When the computational load is well balanced across processors, system noise will be the dominant cause of variation in computation rates.

Noise events vary in duration. As the number of processors grows the probability that during each computational phase at least one processor will expe-

rience the longest noise event approaches one. The longest noise event, though relatively rare, may exceed the nominal length of the computational phase, resulting in greater than 50% performance loss. Here communication times are assumed to be a small fraction of computation times, which is often the case with modern interconnection networks such as Quadrics [3] or Infiniband [2]. Figure 1 shows this graphically: time advances to the right, shaded horizontal boxes represent computation on each node, solid horizontal boxes represent long-duration noise events, and solid vertical boxes represent the communication phases.

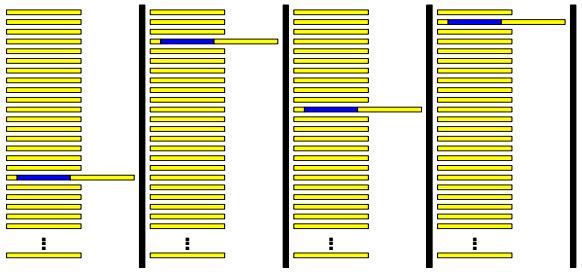


Figure 1: Noise effect

To see how this effect worsens with the number of processors, consider an ideal context with a network with zero latency and a computation in which communication is only synchronization. The computation comprises n processes running on n processors; the processors experience a ~190μs delay every 30ms asynchronously (these are real-world figures drawn from an example given later). Figure 2 shows the effect as a function of the number of processors for 100μs and 1ms computational phases, where latency refers to the delay imposed on each computation/communication cycle. (Many of our applications 1ms computational phases, so this is a realistic value; using 100μs makes problems easier to detect.) The average process latency converges to the maximum delay observed by a single process, such that the application with 100μs computation phase will have approximately one-third the expected performance, the one with 1ms computation phase approximately 80% expected performance.

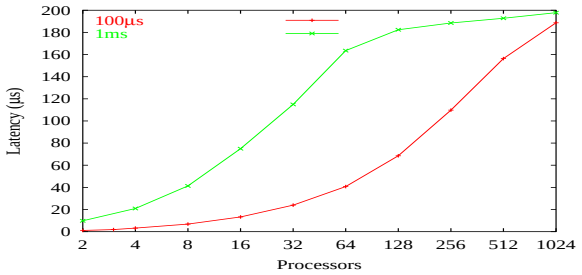


Figure 2: Latency

2 Noise Analysis

System noise is characterized and quantified in three steps:

1. Examinination of microbenchmark behavior;
2. Profiling of the kernel;
3. Quantification of kernel noise.

To measure noise on a single node we use a microbenchmark MicroB. With this benchmark each node's processor executes N iterations of a computation carefully calibrated to run for exactly G μs in absence of noise. If the node is not free of noise some iteration of MicroB will need more than G μs to complete. Table 1 shows the results of running MicroB on one node of our experimental system, a dual AMD Opteron running standard Linux 2.6. Figure 3 shows

G	Ideal Time	Exper. Time	Slowdown
100μs	10 sec	10.16 sec	1.6%
1000μs	100 sec	101.55 sec	1.55%

Table 1: MicroB Slowdown

MicroB time distributions for 1000ms on each node's processor. Figure 3(a) shows that MicroB experiences delays of about 0.5-1.5μs, some for about 8-12μs, and some for about 190μs. Figure 3(b) is similar but there are no 8-12μs delays. Thus there are noise events that occurr only on CPU0; we investigate this first.

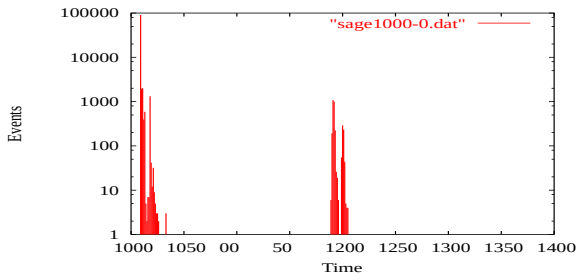
Looking the `/proc/interrupts` proc filesystem we find a strange I/O APIC behavior:

```
           CPU0       CPU1
  0:      99543          0    IO-APIC-edge  timer
  2:          0          0          XT-PIC  cascade
  4:        287          0    IO-APIC-edge  serial
  8:          0          0    IO-APIC-edge  rtc
  9:          0          0   IO-APIC-level  acpi
 15:          2          0    IO-APIC-edge  ide1
 28:      16971          0   IO-APIC-level  eth1
 29:         44          0   IO-APIC-level  ioc0
 30:       5954          0   IO-APIC-level  ioc1
NMI:         26          4
LOC:      97941      98556
ERR:          0
MIS:          0
```

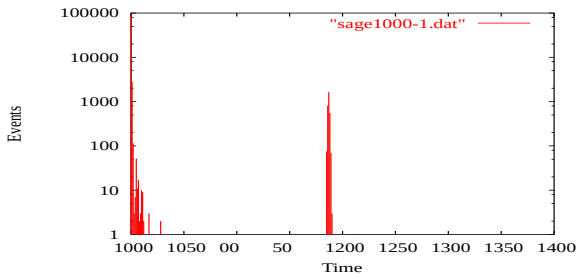
This is a well-known "I/O APIC annoyance" that affects some APIC chips: the I/O APIC sends interrupts only to CPU0. This interrupt controller problem causes unbalanced noise on multi-processor systems.

2.1 Kernel Profiling

The first step in quantifing kernel noise is to understand which and how frequently interrupting functions are called. This may be accomplished using the



(a) CPU 0



(b) CPU 1

Figure 3: MicroB G = 1 ms

kernel profiler OProfile, which is embedded in Linux 2.6, and is capable of profiling all running code with low overhead. Table 2 shows the output of OProfile after MicroB ran for 10 seconds.

The OProfile output shows that the timer interrupt handler is one of the most frequently called functions. The timer interrupt has a frequency of 1000Hz (so one interrupt every 1 millisecond). On SMP systems there is also a local timer for each CPU; it has the same frequency of the global timer (PIT, *Programmable Interrupt Timer*) but it is time shifted.

Every millisecond each CPU receives one interrupt from its local timer and may receive one interrupt from the global timer with a probability of 50%. Taking kernel profiling information and MicroB information together we hypothesize that the timer interrupts and/or the local timer interrupts are responsible for the most frequent noise events—those with delays of about 0.5-1.5μs or 8-12μs, because the most frequently called functions are `timer_interrupt()` and `smp_local_timer_interrupt()`.

2.2 Noise Measurement

The second step is to monitor sources of noise that are identified by the kernel profiling and MicroB tests. We made a kernel modification to get the results shown in Table 3 for interrupts generated by external sources, and in Table 4 for interrupts generated by local timers. This patch includes a proc filesystem interface that can be read to give information about when and how many events occurred.

The timer interrupt is most frequent source and its average delay is about 8-12μs: this matches the noise observed in MicroB tests on CPU0. Moreover, the PIT raises interrupts only on CPU0 because of the I/O APIC, so we have conclusively found the first noise source: the global timer interrupt.

Table 4 shows another source of noise: it has the same frequency of the PIT but introduces a delay of about 0.5-1.5 μs. This noise is present on both CPUs. As illustrated in Table 4, this was caused by the local timer embedded on the CPU chip: it is the second main source of noise in our system. The third most active source of noise is the network interface card (NIC): this network traffic was generated by NFS access.

IRQ	N	Avg	Max	Min	CPU
0	10162	12.237	16.093	8.705	0
...	0	...	...	...	-
28	118	0.751	25.214	0.342	0
...	0	...	...	...	-

Table 3: IRQ noise during MicroB execution

CPU	N	Avg	Max	Min
0	10160	0.576	1.260	0.379
1	10161	0.573	1.205	0.300

Table 4: Local timer noise during MicroB execution

Some interrupt handlers activate tasklets, softIRQs, or workqueues; it is necessary to examine all of these events. Previous results have shown that only few types of interrupts constitute 95% of system noise for a wide variety of UNIX/Linux-based systems. These are the global timer interrupts, local timer interrupts, and network-related interrupts.

Examination of kernel code revealed that

- The main purpose of timer interrupt handler is to check software timer expiration and to update some variables used for time measurement. No softIRQs are activated.
- The main purpose of local timer interrupt handlers is to check for expiration of time quanta allocated for process execution on its CPU, and to update some local statistics variables. The local timer interrupt handler activates softIRQs: the `timer_softirq`.
- The network interrupt handler copies some important information from the NIC to a buffer and then activates a tasklet (there is one tasklet for packets sending and another one for packets receiving).

```
CPU: Hammer, speed 1991.2 MHz (estimated)
Counted CPU_CLK_UNHALTED events with a unit mask of 0x00 (No unit mask) count 100000
samples  %        image name           app name             symbol name
1704     34.8253  vmlinux              microb               timer_interrupt
1120     22.8898  vmlinux              vmlinux              timer_interrupt
451       9.2172  vmlinux              vmlinux              default_idle
61        1.2467  vmlinux              microb               update_process_times
47        0.9606  vmlinux              microb               apic_timer_interrupt
33        0.6744  vmlinux              microb               scheduler_tick
21        0.4292  vmlinux              microb               smp_local_timer_interrupt
15        0.3066  vmlinux              microb               do_softirq
```

Table 2: OProfile output

Table 5 shows results for this analysis. The local timer softIRQ is not very expensive and, in general, it is executed immediately after the local timer interrupt handler. Network send and receive tasklets are normally quick, but can require more time ($> 20\mu s$) to send or receive packets, and so need to be controlled.

Softirq	CPU	N	Avg	Max	Min
LTimer	0/1	20321	0.123	4.758	0.032
NET TX	0	7	0.012	2.138	0.008
NET RX	0	118	0.426	21.521	0.218
ksoftirq	0/1	0	0	0	0

Table 5: Softirq noise during MicroB execution

There remains unidentified noise that occurs every 30ms and introduces a delay of about 180-190μs, for which there seems to be no corresponding kernel activities. Other architectures running the same OS do not exhibit this behavior. It is not possible that this noise is due to an interrupt because of it is present on both CPUs and we have a system with I/O APIC annoyance. We hypothesize that it is caused by by some Hardware feature, likely by some CPU feature. For example, the AMD Opterons have a circuit that reduces processor frequency (by skipping some internal clock cycle) if the temperature of the CPU is too high (Intel P4 have a similar feature). We guess that during MicroB execution this feature is somehow enabled. After some experiments we found that the BIOS was responsible for enabling thermal features on processors. After a BIOS upgrade we have a new MicroB behavior without these noise events.

3 Conclusion

We have shown that in modern supercomputers a significant bottleneck is imposed by the system: traditional performance analysis and profiling tools fail to characterize this problem. Our methodology allows us to clearly identify system noise, quantify the total impact on an application, identify the primary sources, and then eliminate them. The main contribution of is the provision and demonstration of a methodology that is effective for identifying and quantify noise on a system.

References

[1] Erik Hendriks. BProc: The Beowulf Distributed Process Space. In *Proceedings of the 16th Annual ACM International Conference on Supercomputing (ICS'02)*, New York, New York, June 22–26, 2002. Available from `http://www.acl.lanl.gov/cluster/papers/hendriks-ics02/hendriks-ics02.pdf`.

[2] Infiniband Trade Association. Infiniband Specification 1.0a, June 2001. Available from `http://www.infinibandta.org`.

[3] Fabrizio Petrini, Adolfy Hoisie, Wu chun Feng, and Richard Graham. Performance Evaluation of the Quadrics Interconnection Network. In *Workshop on Communication Architecture for Clusters (CAC '01)*, San Francisco, CA, April 2001.

[4] Fabrizio Petrini, Darren Kerbyson, and Scott Pakin. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q. In *Proceedings of SC2003*, Phoenix, Arizona, November 10–16, 2003. Available from `http://www.c3.lanl.gov/~fabrizio/papers/sc03_noise.pdf`.